VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Implementace pluginů pro monitorovací projekt Hyperic HQ

Plugin Implementation for Open Source Project Hyperic HQ

2013 Martin Papež

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Martin Papež**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Implementace pluginů pro monitorovací projekt Hyperic HQ
Plugin Implementation for Open Source Project Hyperic HQ

Zásady pro vypracování:

Cílem práce je seznámení a popis projektu Hyperic HQ a vytvoření několika rozšiřujících pluginů pro monitorování speciálních vlastností.

Práce bude obsahovat:
1. Popis architektury monitorovacího systému Hyperic HQ.
2. Popis vnitřní struktury a způsoby rozšíření jednotlivých částí Hyperic HQ.
3. Implementace pluginu pro monitorování dat v SQL databázi a sledování stavu URL dle vráceného obsahu (velikost, mime type, obsažená klíčová slova, ...).

Seznam doporučené odborné literatury:

[1] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides (Gang of Four): Návrh programů pomocí vzorů. Grada. Praha 2003. ISBN 8024703025
[2] Hyperic Application & System Monitoring [online]. [cit. 2012-10-05]. Dostupné z: http://sourceforge.net/projects/hyperic-hq/
Dále podle pokynů vedoucího bakalářské práce.

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. David Ježek, Ph.D.**

Datum zadání: 16.11.2012
Datum odevzdání: 07.05.2013

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 7. května 2013 .....

Rád bych na tomto místě poděkoval všem, kteří mi pomohli, hlavně mému vedoucímu bakalářské práce Ing. Davidu Ježkovi Ph.D.

Abstrakt

Tato bakalářská práce se zabývá problematikou implementace pluginů pro open source monitorovací systém Hyperic HQ. První část práce je věnovaná teoretickému úvodu do samotného systému Hyperic HQ. V druhá části jsou informace o vnitřních strukturách monitorovacího systému a způsobu komunikace jednotlivých částí. Podrobně je také naznačen postup vytváření pluginu. Následující část se věnuje již konkrétní implementaci pluginů. Je rozdělena na dvě části. Jako první je uveden návrh a implementace pluginu pro monitorování dat v SQL databázi. Následuje návrh a implementace druhého pluginu, který slouží pro sledování stavu URL dle vráceného obsahu.

Klíčová slova: Monitorování systémů, Hyperic HQ, Plugin, Python

Abstract

This bachelor's thesis describes problem of implementation plugins for open source monitoring system Hyperic HQ. First chapter describes theory about Hyperic HQ system. Second chapter describes information about inside structure of system, communication between parts of system. There is also described how to create plugin. Next chapter is dedicated to implementation of plugins and have two parts. First part describes design and implementation of plugin for monitoring SQL data in database. Second part describes design and implementation of plugin, which serves to monitor return content from URL.

Keywords: Monitoring systems, Hyperic HQ, Plugin, Python

Seznam použitých zkratek a symbolů

API	–	Application Programming Interface
CPU	–	Central Processing Unit
DNS	–	Domain Name System
HTML	–	HyperText Markup Language
HTTP	–	Hyper Text Transfer Protocol
IP	–	Internet Protocol
JDBC	–	Java Database Connectivity
JMX	–	Java Management Extensions
POP3	–	Post Office Protocol
RAM	–	Random Access Memory
SNMP	–	Simple Network Management Protocol
SQL	–	Structured Query Language
URL	–	Uniform Resource Locator
VM	–	Virtual Machine
XML	–	Extensible Markup Language
XPath	–	XML Path Language

Obsah

1 Úvod

Tato bakalářská práce se věnuje vytvoření pluginů do open source projektu Hyperic HQ. Samotný projekt Hyperic HQ je monitorovací nástroj, který je rozdělený na dvě části. První částí je administrační server, kde je možné zobrazit si statistiky, ovládat a nastavovat vše potřebné. Druhou částí je agent, který se převážně instaluje na servery, které chceme monitorovat. Agent sbírá a odesílá data do administračního serveru. A tam jsou poté přehledně zobrazeny ve formě grafů.

Hyperic HQ sám o sobě poskytuje velkém množství pluginů, které rozšiřují jeho funkcionalitu. Existují ale případy, kdy je potřeba monitorovat zdroje, které nejsou pokryty žádným dostupným pluginem. V takovémto případě se přímo nabízí možnost naprogramovat si vlastní plugin na míru. Tato možnost je přímo v Hypericu podporována. Nahrání vlastního pluginu na administrační server a jednotlivé agenty je otázkou okamžiku. Stačí si vybrat požadovaný plugin a Hyperic se už o vlastní distribuci postará.

Jako základ pro návrh a implementaci pluginu slouží dokumentace na stránkách projektu a hlavně dokumentace ke komerčnímu projektu vFabric Hyperic 4.6. V těchto materiálech je celkem názorně popsána samotná tvorba pluginu at už s pomocí podpůrných tříd nebo skriptovacího jazyka, popřípadě JMX. Další důležitou částí je vytvoření XML deskriptoru, který se stará o samotné nastavení a reprezentaci metriky.

2 Popis architektury monitorovacího systému Hyperic HQ

2.1 Základní informace

Společnost Hyperic se historicky soustředila na vývoj open source nástroje pro správu IT prostředí. Díky němuž lze efektivně spravovat servery s jakýmkoli běžným operačním systémem, a to v roli aplikačních, webových či databázových serverů. Nyní patří Hyperic do portfolia společnosti VMware a nabízí open source nástroj Hyperic HQ a komerční Hyperic HQ Enterprise [5].

Systém Hyperic HQ slouží pro monitorování a řízení výkonu pro virtuální, fyzickou a cloudovou infrastrukturu. Automaticky dokáže rozpoznat zdroje více než 75 technologií zahrnující vSphere. Sbírá informace o dostupnosti, výkonu a vytváří jejich metriky [2].

Hyperic dovoluje IT manažerům rozšířit klasické monitorovací činnosti k řízení dostupnosti a k zlepšení celkového „zdraví" jejich IT infrastruktury. Ta je účelově určená pro webovou infrastrukturu a navržená tak, že jsou brány v potaz všechny vrstvy infrastruktury včetně hardware, middleware, virtualizace a aplikací. HQ poskytuje systém sledování, trendů a analýzy s jedním kliknutím. Hyperic HQ je první a jediné podnikové open source softwarové řešení pro správu systémů.Tímto umožňuje správci IT technologií spravovat tyto technologie jak nynější, kterou jsou momentálně dostupné na trhu, tak budoucí [4].

2.1.1 Možnosti použití

Systém Hyperic HQ má nepřeberné množství použití. Zde je uvedeno jen několik základních možností:

- automatické prohledání všech komponent virtualizovaných aplikací vSphere
- automatické prohledání, monitorování a řízení softwaru a síťových zdrojů
- monitorování aplikací na jakékoliv platformě včetně Unixu, Linuxu, Windows, Solaris, AIX, HPUX, VMware a Amazon Web Services
- podpora 75 běžných komponent (včetně databází, aplikačních serverů, middlewaru, webových serverů, síťových zařízení a dalších)
- optimalizace pro virtuální prostředí s integrací pro vCenter a vSphere

2.1.2 Základní funkcionalita Hyperic HQ

Hyperic HQ monitoruje a řídí webové aplikace přes širokou škálu platforem a technologií včetně cloudové. HQ poskytuje tuto základní řídící funkcionalitu pro uživatelův software a síťové zdroje:

- **Prohledávání** – HQ agent automaticky prozkoumává softwarové zdroje a naplňuje databázi klíčovými informacemi o dostupném softwaru. Agentem jsou sbírána základní fakta o každém softwarovém zdroji, např. jeho typ, výrobce, verze a umístění.

Obrázek 1: HQ informační panel

Vzhledem k typu zdroje jsou agentem sbírány informace jako je architektura platformy, velikost RAM, rychlost CPU, IP adresa a doménové jméno. Agent používá zabudovaný plugin zdrojů k automatickému prohledání běžně používaných operačních systémů, aplikačních serverů, HTTP serverů, databází, dalšího softwaru a síťových zdrojů. Je možno naprogramovat vlastní plugin zdrojů k řízení softwaru, který HQ nepodporuje a využít také možnosti použití pluginů zdrojů od Hyperic komunity.

- **Organizování** – Zdroje, které agenti HQ objevili jsou uloženy v HQ databázi vzhledem k hierarchickému inventárnímu modelu. Inventární model má smysl pro uspořádání velkého počet softwarových zdrojů a vztahy mezi nimi.

- **Monitorování** – HQ agenti shromažďují metriky, které odrážejí dostupnost, výkon, využití a propustnost. Agenti sbírají standardní soubor metrik pro každý podporovaný typ zdroje. Lze přesně nastavit, jaké metriky je možno shromažďovat ve webovém uživatelském rozhraní a vybrat si, které metriky zobrazit na informačním panelu (obrázek č. 1) nebo domácí stránce v uživatelském rozhraní. Kromě metrik agenti sledují logy, události a změny konfigurace.

- **Ovládání** – HQ lze použít pro vzdálené ovládání a administraci systémových zdrojů. Dostupné ovládací akce závisí na typu zdroje. Na aplikačním serveru je možné provádět úkoly jako spouštění, zastavování a uvolnění zdrojů. Pro databázový server lze vykonat analýzu nebo údržbu.

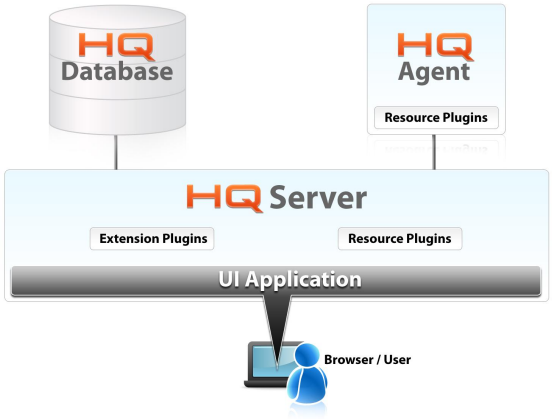
- **Výstrahy, upozornění, stupňování** – Lze nastavit výstrahy na metriky a následné konfigurační akce které HQ provede jakmile se objeví výstraha. V případě objevení výstrahy, může HQ reagovat několika způsoby:
  - zasláním emailového upozornění,
  - nastavením SNMP pasti,
  - komunikací s jinými řídícími systémy.

  Lze rovněž definovat soubor reakcí na výstrahu – eskalační schéma – kdy problémy nejsou propuštěny bez povšimnutí.
- **Prezentace, vizualizace, analýza** – Webové uživatelské rozhraní je vysoce konfigurovatelný nástroj pro výkon a dostupnost. Informační panel (obrázek č. 1) je poskládán z portletů. Portlety lze přidávat, odebírat, přemisťovat a nastavit jaké informace budou zobrazovat [3].

2.2 Rozdělení inventární struktury

- **Aplikace** – kolekce platforem, serverů a služeb organizovaná ke splnění jednoho účelu. Aplikace většinou mají mnoho serverů a služeb. Mohou také běžet na více naž jedné platformě. Typicky J2EE aplikační model v HQ může sestávat z virtuálního hosta Apache, Tomcat Webapps, JBoss sdílených prostředků připojení a instance Oracle.
- **Platforma** – kombinace stroje a operačního sytému nebo jakékoliv síťové či úložné zařízení. Platformy jsou na nejnižší úrovni správy. Zahrnují komponenty jako CPU, síťová rozhraní a souborové systémy.
- **Server** – určitý software, který se instaluje na platformu pod správou.
  - Databáze,
  - middleware,
  - virtualizace,
  - aplikační a webové servery

  jsou všechno příklady serverů. Servery jsou spouštěny na platformách. Platformy hostují několik serverů. Příklad serverů zahrnuje jakoukoliv instalaci JBoss, Tomcat nebo MySQL na dané platformě.
- **Služba** – komponenta serveru vyhrazená ke specifickému účelu. Služby jsou typicky reprezentovány jednotkami práce daného serveru. Různé typy serverů definují seznam jednoho nebo více typů služeb, které poskytují. Jako například:
  - vývoj Webapps v Tomcat,
  - konfiguraci virtuálního hosta v Apache.



Obrázek 2: HQ architektura

Služby mohou být také spojeny přímo s platformou v případě CPU, síťového rozhraní a systému souborů.

- **Typ zdroje** – kategorický popis typu zdroje, který může být spravován. Po instalaci Hyperic HQ popisuje tisíce typů zdrojů, které jsou specifikovány pomocí jména, verze, dostupné metriky a dostupných ovládacích akcí. Typy zdrojů pokrývají platformy, servery a služby.
- **Spravovaný zdroj** – příklad typu zdroje v rámci spravovaného prostředí [4].

2.3 Architektura Hyperic HQ

2.3.1 HQ agent

HQ agent je spuštěn na každém stroji, kde je vyžadována správa od Hyperic HQ. Při prvním spuštění agent automaticky prozkoumá software běžící na daném stroji, a poté periodicky zkoumá změny. HQ agent sbírá informace o dostupnosti, využití, výkonu a propustnosti, provádí logování a zaznamenávání událostí, dovoluje provádět ovládání nad softwarem, například spuštění a ukončení webových aplikačních serverů. Agent odesílá inventární data a metriky, které nasbírá do centrálního HQ serveru.

2.3.2 HQ server a HQ databáze

HQ server přijímá inventární a metrická data od HQ agenta a ukládá je do HQ databáze. Server poskytuje zařízení na správu inventarizace softwaru - implementováno pomocí HQ inventárního a přístupového modelu, seskupení softwarových aktiv užitečným způsobem k usnadnění monitorování a řízení. HQ server provádí detekci výstrah a vykoná

upozorňující a eskalační proces, který je definován. Také provádí akce, které jsou iniciovány přes uživatelské rozhraní nebo API webové služby Hyperic HQ. Server poskytuje autentizaci služeb pomocí vnitřní funkcí nebo externí autentifikační služby.

2.3.3 Uživatelské rozhraní

Hyperic HQ má bohaté webové uživatelské rozhraní. Jeho domácí stránka je HQ informační panel, jednostránkový přehled změn v softwarovém inventáři, problémových zdrojů, posledních výstrahách a metrických grafech pro důležité zdroje. Nad informačním panelem jsou záložky pro zobrazení inventáře, prohlížení a vizualizaci metrik a logiku výstrah.

2.3.4 HQ API

HQ API je webová služba, která poskytuje programový přístup ke všem datům a funkcionalitě na HQ serveru. Toto je užitečné pro zefektivnění běžných implementací a konfigurací, které Hyperic provádí. Například je možné za pomoci příkazové řádky provádět hromadné aktualizace a konfigurace. HQ API také dovoluje implementovat rozhraní mezi Hyperic a jinými řídícími systémy. Je možné napsat nástroj, který pomocí volání API extrahuje inventární data a připraví je pro import do systémů sledování majetku.

2.3.5 Pluginy

Je možné rozšířit schopnosti Hyperic HQ dvěma typy pluginů:

- HQ agent používá zdrojové pluginy pro prozkoumání, monitorování a ovládání softwarových zdrojů. HQ má mnoho zabudovaných zdrojových pluginů. Je možnost naprogramovat si vlastní plugin zdroje, který HQ nepodporuje.
- Je možné naprogramovat rozšiřující plugin pro přidání možností HQ uživatelského rozhraní, skripty pro automatizaci běžně prováděných postupů a rozhraní pro webové služby dalších řídících systémů [3].

3 Popis vnitřní struktury a způsoby rozšíření jednotlivých částí systému Hyperic HQ

3.1 Základní údaje

Hyperic poskytuje proaktivní řízení výkonu s úplnou a trvalou viditelností do aplikací a infrastruktury. Produkuje více než 50 000 výkonnostních metrik na více než 75 technologiích v každé vrstvě.

Hlavní komponenty Hypericu zahrnují Hyperic server, agent, databáze a Hyperic uživatelské rozhraní také známé jako Hyperic Portál.

3.1.1 Hyperic Portál

Uživatelské rozhraní Hyperic Portál se dá lehce nakonfigurovat, rozšířit pro monitorování a analýzu výkonnosti a dostupnosti. Hlavní funkce uživatelského rozhraní zahrnují:

- **Informační panel**. Informační panel je první stránka která je zobrazena po spuštění uživatelského rozhraní. Informační panel obsahuje několik portletů – Poslední výstrahy, Dostupnost, Problémové zdroje a další. Na každém portletu je zobrazen souhrn událostí a zdrojů zájmů. Je možné nastavit informační panel pro každého uživatele zvláště. Lze odebrat a přeorganizovat portlety na informačním panelu a nastavit jaká data bude portlet zobrazovat.
- **Rozcestník zdrojů**. Je možné přejít na záložku zdrojů často nazývanou také rozcestníkem zdrojů k vyhledání požadovaného zdroje, zobrazení možností zdrojů, zobrazení metrických dat a grafů a také zahájit řízení zdrojů. Administrátoři Hyperic používají rozcestník zdrojů pro nastavení monitorování a definice výstrah na zdrojích.
- **Globální monitorovací zobrazení**. Uživatelské rozhraní Hyperic obsahuje následující stránky, které prezentují výsledky monitorování zdrojů:
  - **Centrum Operací**. Celkový pohled zahrnující výstrahy, události a současné odpojené zdroje.
  - **Centrum Výstrah**. Pohled na výstrahy a jejich definici.
  - **Centrum Událostí**. Pohled na logování událostí, konfigurační události a výstrahy.
  - **Nagios data**. K dispozici v nasazení, které mají integrovaný Hyperic s Nagios.
  - **Momentálně odpojen**. Seznam momentálně nedostupných zdrojů.
- **Pohled na typy zdrojů**. Některé stránky v uživatelském rozhraní Hyperic jsou specifické pro konkrétní typ zdroje, například vSphare pohled pro vCenter a vCenter spravované zdroje a GemFire pro monitorování částí vFabric GemFire distribuované mezipaměti prostředí.

3.1.2 Základní fakta pro nové uživatele

Zdroje, které se uživateli zobrazují a úroveň oprávnění se řídí danou rolí.

Některé zdroje je nutné nakonfigurovat pro monitorování. Ačkoliv Hyperic začne monitorovat většinu zdrojů jakmile jsou přidány do inventáře, určité typy zdrojů je potřeba nakonfigurovat, aby agent mohl sbírat metriky. Například JMX URL adresa a přístupové údaje musí být zadány v Hypericu pro to, aby byl agent schopný monitorovat skrz JMX. Metriky shromážděné pro zdroj se řídí výchozím nastavením pro metrické kolekce typu prostředku.

3.2 Platformy, Servery a Služby

Každá zdrojová instance v Hypericu má jeden z následujících inventárních typů: platforma, server nebo služba.

3.2.1 Hierarchie Platforma-Server-Služba

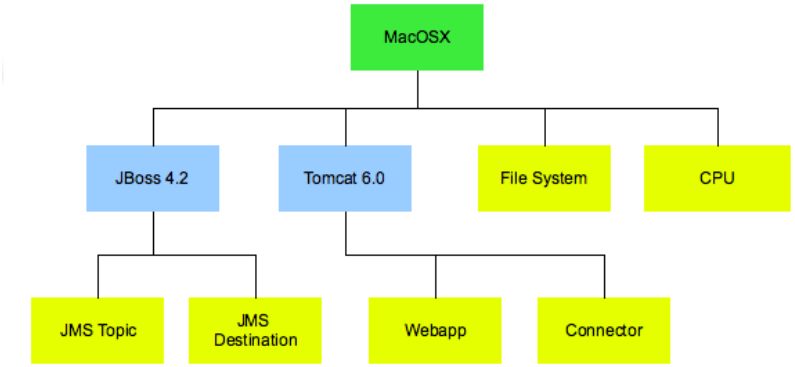
Platforma, server a služba jsou vy Hypericu hierarchicky propojeny.

- **Platforma** je převážně stroj s operačním systémem, na kterém je spuštěn HQ agent. Jsou také platformní typy pro virtuální a sítové hosty.
- **Server** je softwarový produkt, který je spuštěn na platformě.
- **Služba** je zdroj, který je nedílnou součástí nebo běží na platformě či serveru. Je jedno jestli je služba svázána s platformou nebo serverem, pro Hyperic je to služba. Ačkoliv služby asociované s platformou jsou většinou považovány za platformní službu.

Hyperic automaticky prozkoumá většinu platforem, serverů a služeb a naplní Hyperic databázi klíčovými informacemi o každé objevené položce a jejím vztahu s ostatními zdroji.

Na obrázku 3 je zobrazena konkrétní hierarchie platforma-server-služba. (Jenom některé servery a služby jsou zobrazeny z celkové hierarchie.) Nápis na každém zdroji naznačuje jeho typ. Hierarchie obsahuje následující:

- platformu jejíž typ je „MaxOSX“
- dvě platformní služby, jejichž typy jsou „Souborový systém“ a „CPU“
- dva servery, jejichž typy jsou „JBoss 4.2“ a „Tomcat 6.0“
- čtyři služby, které jsou spuštěny na serverech, jejichž typy jsou „JMS Topic“, „JMS Destination“, „Webapp“ a „Connector“



Obrázek 3: Hierarchie Platforma-Server-Služba

3.2.2 Platformy

V Hypericu jsou dva hlavní druhy platforem:

- **Platformy operačních systémů**. Platforma operačního systému je počítač a operační systém, který je na něm spuštěn. Hyperic automaticky prozkoumá platformu operačního systému pomocí systémového pluginu Hyperic. Není možné přidat manuálně operační systém do inventáře. Hyperic podporuje následující typy platforem operačních systémů:
  - AIX
  - FreeBSD
  - HPUX
  - Linux
  - MacOSX
  - Solaris
  - Unix
  - Win32
- **Virtuální a síťové platformy**. Hyperic podporuje různé typy platforem které nesouvisí s konkrétním fyzickým strojem a tradičním operačním systémem. Tyto jsou:
  - Zdroje, které Hyperic agent monitoruje vzdáleně přes sít, jako je sítový host a zařízení,
  - virtuální zdroje, jako je VMware vSphere host a VM
  - a distribuovaný soubor zdrojů jako GemFire distribuované systémy.

Hyperic agent nepodporuje automatické prozkoumání a monitorování virtuálních a sítových platforem. Typicky se tyto platformy vytvářejí manuálně (použitím příkazu Nová platforma), nebo přinejmenším poskytnou nastavení dat pro zdroje, které dovolí agentovi, aby je spravoval. Toto jsou virtuální a sítové platformy, které Hyperic podporuje:

- Cisco IOS
- Cisco Pixos
- GemFire Distributed System
- NetApp Filer
- Network Device
- Network Host
- VMware vSphere Host
- VMware vSphere VM
- Xen Host

3.2.3 Servery

V Hypericu je server softwarový produkt, který je spuštěn na platformě. Servery poskytují komunikační rozhraní a provádějí specifické úkoly na základě požadavků. Příklady serverů jsou TomCat, JBoss a Exchange. Většina typů severů jsou automaticky objeveny pomocí serverových Hyperic pluginů. Jestliže plugin, který spravuje server nepodporuje automatické prozkoumání, nebo když automatické prozkoumání selže, je možné manuálně vytvořit server.

3.2.4 Služby a platformní služby

Služba je softwarová komponenta v Hypericu, která se věnuje konkrétnímu úkolu, který je prováděn na serveru nebo platformě. Služba, která je spuštěna na serveru je označovaná jako služba. Zatímco služba, která je spuštěna na platformě je označována jako platformní služba. Zdrojový plugin, který má na starosti prozkoumání platformy nebo serveru také prozkoumá klíčové služby – jako je CPU, síťová rozhraní, souborový systém a další – běžící na platformě.

Kromě toho může autorizovaný uživatel jasně nakonfigurovat platformní služby na platformě, které slouží jako proxy pro zdroje, které Hyperic agent může monitorovat přes síť, například DNS nebo POP3 službu [6].

3.3 Pluginy

Pluginy jsou rozhraní mezi Hyperic HQ a produkty na síťi, které je možné spravovat. Hyperic dokáže detekovat tisíce produktů díky jeho standardním pluginům, ale je také

možné rozšířit funkcionalitu Hypericu o produkty, nebo části produktů, které nejsou zatím pokryty, za pomoci vlastních pluginů.

Vývoj pluginů vyžaduje pochopení inventárního modelu HQ popsaného zdroji, typy zdrojů a inventárními typy a funkcemi, které plugin implementuje.

Je možné pluginy použít pro prohledávání, sběr dat a ovládání zdrojů. Pluginy nemohou být ale použity na změnu výstrah, reportování nebo podobnou serverovou funkcionalitu.

Pluginy v Hyperic HQ jsou samostatné .jar nebo .xml soubory, které jsou nasazeny na server a každého agenta, který plugin spouští. Každý plugin minimálně obsahuje popis v XML, který je uložen buď samostatně nebo zabudovaný do .jar souboru.

3.3.1 Role serveru a agenta v pluginech

Je potřeba plugin nasadit, jak na server, tak současně na agenta. Server a agent zastupují každý různé role v souvislosti s pluginy:

Agent shromažďuje všechna data ze zdrojů a všeobecně komunikuje se zdroji. Pomocí pluginu agent může:

- Automaticky objevit zdroje.
- Shromažďovat metriky ze zdroje.
- Provádět podporované ovládací akce.

Server se zabývá meta-daty, což znamená, že ví o:

- Platformě, serveru a typech služeb zdrojů, a jak plugin mapuje cílené zdroje do inventárního modelu.
- Konfiguračním schématu pro každý zdroj.
- Zobrazování dat od zdrojů a metrických dat v uživatelském rozhraní Hypericu.
- Definici kontrolních akcí.

3.3.2 Implementace pluginů

Pluginy mohou být vytvořeny pro různé druhy zdrojů. V závislosti na druhu zdroje, jak komunikuje a poskytuje data mohou být napsány různé typy implementace pluginů. Tyto typy jsou následující:

- Script
- JMX
- SQL
- SNMP

3.3.3 Použití podpůrných tříd ke zjednodušení pluginů

Hyperic HQ poskytuje mnoho podpůrných tříd (jsou to pluginy), na které je možné se odvolat v rámci vlastních pluginů a zjednodušit si implementaci. Hyperic HQ poskytuje následující podpůrné třídy (tabulka č 1):

Kategorie	**Podpůrné třídy**	**Kdy je možno tuto třídu použít**
Skriptování	qmail, Sendmail, Sybase	
SNMP	Squid, Cisco IOS	
JMX	JBoss, WLS, WAS, ActiveMQ, Jetty	
JDBC	MySQL, PostgreSQL, Oracle	K shromažďování metrik z databázových tabulek
Win-Perf Counters	IIS, Exchange, DS, .NET	K shromažďování metrik aplikací používající perf counter
SIGAR	System, Process, Netstat	Pro komunikaci s operačním systémem. SIGAR je proprietární API nezávislé na OS
Síťové protokoly	HTTP, FTP, SMTP, a další	Pro komunikaci s platformními službami, které má HQ zabudované, ale je potřeba shromažďovat další metriky od nich
Vendor	Citrix, DB2, VMware	

Tabulka 1: Podpůrné třídy v Hyperic HQ a jejich využití

Rozhraní HQ pluginů je jednoduché. Například měřící HQ plugin má jenom jednu metodu (getValue). Autoinventární plugin má jenom jednu metodu pro každou úroveň v inventáři. Nejtěžší část při psaní pluginu je rozmyslet si:

- Jak je možné dostat data ze spravovaného zdroje?
- Kde mají být tyto data umístěna v hierarchii inventárního modelu? Na jaké inventární úrovni (platforma, server, služba)?

3.4 XML Deskriptor

Deskriptor pluginu je XML soubor, který je definovaný jako co může plugin dělat a jak to má dělat - typy zdrojů, které spravuje a pro každý typ jakou funkci vykonává, data zdrojů, které požaduje a prozkoumává a metriky, které vrací. Každý plugin má

soubor deskriptoru. Plugin, který používá podpůrné třídy Hyperic nebo skript který vykonání funkce správy potřebuje také svůj deskriptor soubor. Deskriptor pro plugin, který používá vlastní řídící třídy je zabalen se třídami v souboru JAR pro nasazení.

XML deskriptor je v podstatě plugin, který obsahuje informace o tom jakým způsobem je plugin realizovaná. V případě skriptu, musí být v deskriptoru uvedena cesta ke skriptu, který se má spouštět.

3.4.1 Nahrání pluginu do systému Hyperic HQ

K použití pluginu je potřeba vlastní plugin nejdříve nahrát na server, který provede automatickou distribuci jednotlivým agentům. Toto je možné z uživatelského rozhraní Hyperic HQ. Nad informačním panelem je záložka `Administration`. Po jejím zobrazení je potřeba ještě kliknout na na odkaz `Plugin Manager` v sekci `HQ Server Settings`. Zobrazí se stránka se všemi nainstalovanými pluginy a dalšími informacemi o nich. Pomocí tlačítka `Add/UpdatePlugin(s)` v pravé spodní části stránky se provádí samotné nahrání pluginu. V dialogovém okně je potřeba vybrat soubor, který obsahuje plugin a ten pomocí tlačítka `Upload` nahrát do systému.

3.4.2 Hierarchie spravovaných typů zdrojů

Deskriptor pluginu definuje každý zdrojový typ, který bude spravovat, v některých případech jeden typ, ale častěji hierarchii typů. Například server (Tomcat 6.0) a jeho služby (Vhosts). Plugin může spravovat několik typů zdrojových hierarchií, například Tomcat plugin v Hypericu spravuje Tomcat 5.5 a také Tomcat 6. Deskriptor pro takovýto plugin definuje hierarchii pro každou z verzí.

Ačkoliv plugin může spravovat platformu a jednu nebo více úrovní závislých zdrojů, v praxi prakticky všechny zdroje na platformní úrovni jsou spravovány jedním Hyperic pluginem - systémovým pluginem (system-plugin.jar). Systémový plugin prozkoumává a spravuje všechny podporované platformy operačních systémů (Unix, Linux, Win32, Solaris, MacOSX, AIX, HPUX, a FreeBSD) a platformní služby (jako je sítové rozhraní, CPU, souborový systém).

Jediné další pluginy Hypericu, jenž spravují zdroje, které Hyperic považuje za platformy jsou ty, jenž spravují virtuální nebo sítové hosty.

3.4.3 Shromažďované metriky pro každý typ zdroje

Deskriptor specifikuje každou metriku, kterou plugin získává pro každý typ zdroje, který spravuje. Například plugin Tomcat získává „Dostupnosť“, „Momentální počet vláken“ a „Momentální zaneprázdněná vlákna“ pro službu „Vlákna“. Pravidla pro získávání metrik jsou definovaná ve strukturovaném výrazu definovaném jako šablona metriky. Šalona metriky identifikuje cílovou metriku podle jména návratové související měrné třídy a poskytuje data, která třída požaduje k získání metriky (např. zdrojový JMX ObectName).

3.4.4 Kostra deskriptoru

Struktura deskriptoru pluginu je stejná jako hierarchie zdrojových typů, které plugin spravuje, vyjádřená podmínkami inventárního modelu Hyperic. Deskriptor pluginu obsahuje element typu zdroje - `<platform>`, `<server>` nebo `<service>` - pro každý typ zdroje, který je spravován. Hierarchie zdrojových elementů v deskriptoru musí odrážet vztahy mezi spravovanými typy zdrojů. Například element `<server>` pro Tomcat obsahuje (je rodičem) element `<service>` pro typ Vhost.

Typický příklad hierarchie deskriptoru:

```
<plugin> (root)
    <server>
        <service>
    <server>
    <service>
```

3.4.5 Fakta o šabloně metrik

Každou metriku, kterou plugin shromažďuje má šablonu metriky, která vyjadřuje požadavek na specifickou metriku pro specifický zdroj ve formátu kterému Hyperic agent rozumí. Šablona metriky má formu: `Domain:Properties:Metric:Connection`

Šablona metriky poskytuje informace pro měrnou třídu, které potřebuje k získání metriky. Tyto informace zahrnují parametry zdroje (data spojení, typ zdroje a jména a tak dále) a parametry metriky (kategorie, jednotky měření, jestli je to indikátor, a tak dále). Tyto zdrojové a metrické parametry jsou definovány v elementech `<option>`, `<property>` a `<metric>` pro typ zdroje. Šablona zdroje sjednocuje tato data pro konkrétní metriku jako strukturovaný „požadavek metriky“, který Hyperic agent dokáže splnit.

Plugin Hypericu spravující hierarchii serverových služeb často sbírá přes tisíc jedinečných metrik. K usnadnění procesu definice šablon metrik, vývojáři pluginů typicky definují šablonu z hlediska proměnné, která vrací hodnoty relevantní s `<option>`, `<property>` a `<metric>` parametry [7].

4 Implementace pluginů pro monitorování

Existuje mnoho pluginů pro monitorování, ale žádný nesplňuje daná kritéria pro monitorování přibývajících dat v tabulkách databázích nebo pro monitorování stavu URL dle vráceného obsahu, proto je návrh a implementace těchto pluginů cílem této práce.

Plugin pro monitorování dat v databázových tabulkách bude implementován pomocí podpůrných tříd Hyperic. A plugin, který monitoruje stav URL dle vráceného obsahu bude implementován pomocí skriptu, aby bylo možné porovnat oba dva přístupy k tvorbě pluginů.

4.1 Plugin pro monitorování dat v SQL databázi

Plugin pro monitorování dat v SQL databázi je navržen tak, že používá tři druhy metrik. První druh je vždy vyžadovaný u Hyperic pluginů a musí být bezpodmínečně přítomen, jedná se o dostupnost služby. Tato dostupnost se může zjišťovat různým způsobem. Je důležité, ale aby vždy dokázala vrátit 0 nebo 1. Podle toho se vlastně vyhodnocuje, jestli je služba dostupná nebo nedostupná. Poté se tato metrika zobrazuje v uživatelském rozhraní Hyperic jako procentuální hodnota (např. 85% dostupnost). Dá se podle toho velice dobře určit v jakém je služba stavu, a jak často u ní dochází k výpadku.

Druhá metrika spočívá v SQL dotazu do databáze, který zjišťuje celkový počet záznamů v tabulce. Tímto se dá celkem snadno zjistit, jestli data do tabulky přibývají nebo se nemění. Hodnota je celočíselná a zobrazuje se poté v uživatelském rozhraní Hyperic a generují se pomocí ní grafy, kde je vidět přírůstek dat.

Poslední metrika je navržena tak, aby dokázala z tabulky podle jména sloupce odečíst poslední hodnotu typu `datetime` a tuto hodnotu vrátit. Toto je velice důležité pro tabulky, které si zaznamenávají datum a čas uložení záznamů do sloupců. Je potom možné si tyto data z databáze vyvolat a nechat si je pohodlně zobrazit a zjistit, jestli data přibývají.

Plugin je možné použít pro sledování přibývání dat do tabulky a s pomocí nastavení různých výstrah se nechat informovat o požadovaném stavu. Takto je možné docílit například toho, že při neplnění tabulky daty se po určité době odešle email s informacemi o neplnění na požadovanou emailovou adresu.

4.1.1 Návrh řešení

Nejdříve bylo potřeba stanovit, jakým způsobem zjišťovat, jestli data v tabulkách přibývají. Toto je možné zjistit pomocí dotazu, který vrátí celkový počet záznamů z tabulky (např. `SELECT * FROM tabulka;`). Tím zajistíme, že se nám do metriky dostane celkový počet záznamů z tabulky. Toto je základní přístup k problému. Jestliže jsou v tabulce sloupce, které mají typ `datetime` a do nich se zaznamenává datum a čas, kdy se do tabulky záznam uložili, je možné tyto sloupce monitorovat a zjistit si poslední zadaný datum a čas a tento údaj vrátit jako metriku. Toto je možné provádět SQL příkazem, který dokáže vyfiltrovat daný sloupec a na něm provést řadící funkci a vybrat poslední záznam.

Bylo potřeba také vyřešit správu připojení do databáze. Hyperic má však pro tento případ přichystané podpůrné třídy, které je možné využit pro různé implementace pluginů. Tyto třídy se hodí pro různé použití a je dobré je využít, protože dokáží usnadnit některé úkoly, které by trvali příliš dlouho. V mém případě byla použita jenom jedna. Třída `org.hyperic.hq.plugin.sqlquery.SQLQueryMeasurementPlugin` se stará o spojení s databází a prováděním SQL dotazů. Pro její použití je nutné specifikovat jaký se použije JDBC driver. Tyto drivery jsou implementovány téměř pro každou databázi. V pluginech jsou použity tyto tři drivery:

- **MySQL:** `com.mysql.jdbc.Driver`
- **MSSQL:** `com.microsoft.sqlserver.jdbc.SQLServerDriver`
- **PostgreSQL:** `org.postgresql.Driver`

Dále je potřebu uvést připojovací url adresu databáze používající vlastní JDBC formát. MySQL používá následující formát: `jdbc:mysql://localhost/jmenoDatabaze`. Nutné jsou také přihlašovací údaje do databáze, jako je jméno a heslo.

Při návrhu pluginu byla brána v poraz rozdílnost jednotlivých databázových produktů a jejich databází, proto jsem implementoval plugin pro následující databáze: MySQL, MSSQL a PostgreSQL. Pro každý typ databáze je implementován samostatný plugin, který se starý o sběr dané metriky. Bylo potřeba takto pluginy rozdělit, protože každý pracuje jinak s typem `datetime`, který je nutné ještě pro měření metrik převést na typ `timestamp`. Tento se poté zobrazuje v uživatelském rozhraní Hyperc. Kde je možné sledovat jeho růst.

Pro tento typ pluginu je požit pouze jeden XML soubor, který obsahuje jak samotný deskriptor, tak vlastní implementaci. Tímto bylo docíleno výrazného zjednodušení pluginu. Nebylo by to však možné, bez použití podpůrné třídy pro práci s SQL, kterou Hyperic poskytuje.

4.1.2 Vlastní implementace

Implementace probíhala pro každý typ databáze samostatně. Zde bude ukázána implementace pro databázi MySQL. Implementace pro ostatní databáze je obdobná.

Plugin growth_of_table_mysql obsahuje dvě platformní služby, které se zobrazují v uživatelském rozhraní jako *NumOfRec_MySQL* což je zkrácený název pro počet záznamů a *LastRecInCol_MySQL* což je zkrácený název pro poslední záznam ve sloupci. Každá služba má svojí jedinečnou metriku a svoje nastavení.

Služba *NumOfRec_MySQL* je implementována pomocí podpůrné třídy, které nabízí zjednodušenou práci s SQL dotazy pomocí zápisu do atributu `template`. Syntaxe příkazu je následující: `sql:Query:MetricName`. Konkrétní příklad pro zjištění dostupnosti služby je: `sql:SELECT COUNT(*) FROM table WHERE 1=1:Availability`. Tento příkaz je nutné zadat do ohraničeného elementu `metric` v XML deskriptoru pluginu. Konkrétní příklad pluginu je na výpisu č. 1.

```
<plugin>
    <property name="PLUGIN_VERSION" value="4.6.6"/>
    <property name="PLUGIN_NAME" value="growth_of_table_mysql"/>

    <service name="NumOfRec_MySQL">
        <config>
            <option name="jdbcDriver" type="hidden"
            description="JDBC Driver Class Name"
            default="com.mysql.jdbc.Driver"/>

            <option name="jdbcUrl"
            description="JDBC Connection URL"
            default="jdbc:mysql://localhost/databaseName"/>

            <option name="jdbcUser"
            description="JDBC User"/>

            <option name="jdbcPassword" type="secret"
            optional="true"
            description="JDBC Password"/>

            <option name="tableName"
            description="Name of desired table"/>
        </config>

        <metric name="Availability"
        template="sql:SELECT COUNT(*) FROM %tableName% WHERE 1=1:${name}"
        indicator="true"/>

        <metric name="Number of records"
        template="sql:SELECT COUNT(*) FROM %tableName%:${name}"
        indicator="true"
        collectionType="dynamic"
        interval="6000"/>
    </service>
</plugin>
```

Výpis 1: XML plugin pro sběr dat z databázových tabulek

Jednotlivé metriky jsou v XML deskriptoru popsány na následujících stránkách.

```
<metric name="Availability"
template="sql:SELECT COUNT(*) FROM %tableName% WHERE 1=1:${name}"
indicator="true"/>
```

V příkladu je uveden název metriky `name="Availability"` vlastní šablona `template` a indikátor, zda se bude zobrazovat v uživatelském rozhraní `indicator="true"`.

Metrika pro vlastní měření, jestli do tabulky přibývají data pomocí zjištění celkového počtu záznamů v tabulce je následující:

```
<metric name="Number of records"
template="sql:SELECT COUNT(*) FROM %tableName%:${name}"
```

```
indicator="true"
collectionType="dynamic"
interval="6000"/>
```

Významy jednotlivých atributů jsou stejné jako v předchozím případě, ale jsou zde některé navíc. Atribut `collectionType="dynamic"` značí, že se metrika může libovolně měnit, stoupat i klesat. Toto může být trochu zavádějící, ale v tomto konkrétním případu je nutní mít nastaven takovýto typ, aby se v grafech v uživatelském rozhraní objevili požadované počty záznamů. Atribut `interval="6000"` značí, jak často se bude metrika zaznamenávat, což je v tomto případě 6000 ms.

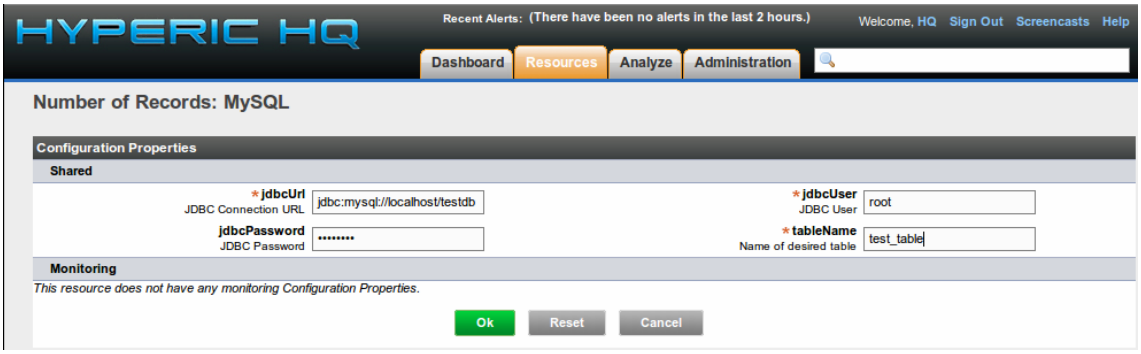
Každý plugin obsahuje konfigurační element `<config>` do kterého se zapisuje nastavení pluginu, které se potom objevuje v uživatelském rozhraní na stránce vytváření monitorovací služby. Zápis elementu `<config>` pro nastavení služby je následující:

```
<config>
    <option name="jdbcDriver" type="hidden"
    description="JDBC Driver Class Name"
    default="com.mysql.jdbc.Driver"/>

    <option name="jdbcUrl"
    description="JDBC Connection URL"
    default="jdbc:mysql://localhost/databaseName"/>

    <option name="jdbcUser"
    description="JDBC User"/>

    <option name="jdbcPassword" type="secret"
    optional="true"
    description="JDBC Password"/>

    <option name="tableName"
    description="Name of desired table"/>
</config>
```

Element `<option>` obsahuje například atributy `name="jdbcDriver"`, kde je uloženo jméno, `type="hidden"`, který říká, že tato možnost se nemá zobrazovat v uživatelském rozhraní Hyperic, `description="JDBC Driver Class Name"`, kde je uveden popis položky a `default="com.mysql.jdbc.Driver"` který říká, jak se má nastavit výchozí hodnota. Jak vypadá konfigurační element zobrazený v uživatelském prostředí Hyperic je vidět na obrázku 4.

Toto všechno je uloženo v XML deskriptoru a obaleno elementem `<service>`, který říká, že se jedná o plugin pro službu.

Služba *LastRecInCol_MySQL* je implementována podobně jako předchozí služba s tím rozdílem, že byl použit jiný atribut `template` pro získání informace o poslední hodnotě zadaného sloupce, který má následující podobu:



Obrázek 4: Konfigurační formulář služby NumOfRec_MySQL

```
template="sql:SELECT UNIX_TIMESTAMP(%columnName%)
FROM %tableName%
order by %columnName% desc limit 1:${name}"
```

Tento příkaz provede výběr daného sloupce seřazeného podle data a získání poslední časové hodnoty, která je následně převedena na typ `timestamp`. Pro tuto službu je ještě potřeba nastavit, jaký sloupce z tabulky chceme sledovat.

4.1.3 Otestování pluginu

Otestování pluginů probíhalo na vlastní instalaci Hyperic HQ s agentem nainstalovaným na tom samém stroji. Byla vytvořena databáze `testdb` a v ní byla vytvořena tabulka `test_table`, do které byla vytvořena inicializační data. Tabulka `test_table` se skládá ze tří sloupců. První sloupec se jmenuje `id`, je typu `int` a je v něm uloženo id záznamu, které se automaticky doplňuje po přidání záznamu do tabulky. Další sloupec se jmenuje `name`, je typu `varchar` a je zde uloženo jméno záznamu. Poslední sloupec se jmenuje `date`, je typu `datetime` a je v něm uložena informace o čase a datu přidání záznamu do tabulky.

Na začátku byly do databáze uloženy čtyři záznamy s jedinečným `id` a vlastním časem uložení ve sloupci `date`. Poté bylo provedeno nastavené pluginů na služby *NumOfRec_MySQL* a *LastRecInCol_MySQL* s nastavením této konkrétní databáze a její tabulky. V případě služby *LastRecInCol_MySQL* byl ještě nastaven sloupec `date` pro monitorování času a data.

Postupně se tabulka plnila daty a kontroloval se její stav pomocí pluginů. Jak se měnili grafy v průběhu testování je možné si prohlédnout na obrázku č. 5. Z obrázku je patrné jak služba pro sběr celkového počtu záznamů vykazuje růst, když jsou záznamy do databáze přidávány. U služby která zobrazuje čas posledního záznamu je patrný nelineární a skokový růst. To je způsobeno tím, že záznamy byli do databáze vkládány s různým datem, ne vždy bezprostředně za sebou následujícím.

Testovací data jsou znázorněna v tabule č. 2

Obrázek 5: Ukázka grafů monitorovací služby s přibývající dobou a dalšími záznamy

4.2 Pluginu pro sledování stavu URL dle vráceného obsahu

Tento plugin obsahuje celkem sedm metrik pro rozličné použití. Jako základní je zde metrika dostupnosti, které je vyžadovaná u každého pluginu. Tato metrika určuje, jestli je služba v provozu nebo není. Jedná se o metriky, které se zjišťují pomocí http dotazu. Je tedy potřeba zjišťovat návratový kód tohoto dotazu, proto je to jedna z metrik. Plugin také obsahuje metriku pro zjištění velkosti obsahu, která udává velikost odpovědi v Byte. Pro správnou funkcionalitu následujících metrik je třeba provést dodatečná nastavení v uživatelském rozhraní pro upřesnění metriky. Dodatečné metriky jsou:

- Typ obsahu (Content-Type)
- Klíčové slovo v HTML tagu `<meta name="keywords"/>` (Keyword status)
- Počet nalezených elementů pomocí XPath (Number of elements by XPath)
- Vyhledání řetězce ve viditelném HTML (String found)

Typ obsahu je metrika, pro kterou je potřeba zadat, jaký typ má vyhledávat v HTTP odpovědi. Jako příklad je možné uvést Content-Type: `text/html`. Toto je základní typ, který se zobrazí po načtené běžné HTML stránky. Typů je velké množství a tato metrika počítá s tím, že je bude uživatel nastavovat. Je možné sledovat, jestli se zobrazují požadované obrázky pomocí Content-Type: `image/png`. V případě, že je typ obsahu v odpovědi nalezen do metriky se uvede číslo 1, v opačném případě se uvede číslo 0.

id	name	date
1	a	2013-04-05 13:20:00
2	b	2013-04-15 13:20:00
3	c	2013-04-25 13:20:00
..	.....	.....
10	j	2013-06-05 13:20:00
11	k	2013-06-06 13:20:00
12	l	2013-06-07 13:20:00

Tabulka 2: Testovací data v tabulce test_table

Klíčové slovo v HTML tagu `<meta name="keywords"/>+` je metrika pro vyhledávání klíčových slov v HTML stránce. Tento HTML tag slouží pro vyhledávače a uvádějí se zde klíčová slova pro zobrazenou stránku z důvodu kategorizace záznamů. Hledá se vždy jenom jedno klíčové slovo. V případě, že je požadované slovo nalezeno do metriky se uvede číslo 1, v opačném případě se uvede číslo 0.

Počet nalezených elementů pomocí XPath slouží pro vyhledávání v HTML pomocí XPath a vrácení počtu nalezených elementů. Je možné zadávat jednoduché zápisy pomocí XPath, ale není problém vytvořit i složitější konstrukce, pro nalezení požadovaných elementů, které mají více atributů a další parametry. Jako příklad lze uvést následující XPath: `//div[starts-with(@id, "photo_") and @class="new"]`, který prohledává celý dokument a hledá element `div`, jenž má v sobě atribut `id`, jehož hodnota začíná na `photo_` a zároveň obsahuje atribut `class`, který má hodnotu `new`.

Vyhledání řetězce ve viditelném HTML je metrika, kterou je možno použít na hledání textu v HTML, které se zobrazí v prohlížeči. To znamená, že z hledání jsou vypuštěny všechny HTML tagy, ponechán je pouze textový obsah. Jestliže je hledané slovo nalezeno do metriky se uvede číslo 1, jinak se uvede číslo 0.

4.2.1 Vlastní implementace

Plugin je implementován ve skriptovacím jazyce Python, který byl použit pro jeho velké rozšíření, čitelnost kódu a mnoho podpůrných knihoven. V jazyce Python je napsán skript, který se volá pomocí samotného pluginu. Plugin obsahuje deskriptor, kde je popsáno, která metrika používá jaký příkaz. Samotná skript se skládá z jedné metody, která se připojuje na zadanou url a pomocí podpůrných metod sbírá a filtruje požadovaná data. Skript je implementován následovně:

```
import sys
import urllib2
import lxml.html

def getMetric() :

    code = 0
    contentTypeStatus = 0
```

```
keywordStatus = 0
responseCode = 0
contentLength = 0
xpathElements = 0
stringToFindStatus = 0

# Prirazeni vstupnich parametru
url = sys.argv[1][3:]
contentType = sys.argv[2][3:]
keyword = sys.argv[3][3:]
xpath = sys.argv[4][3:]
stringToFind = sys.argv[5][3:]

request = urllib2.urlopen(url)

# Response code
responseCode = request.getcode()

info = request.info()
# Content-Length
contentLength = info.get("Content-Length")

# Content-Type
if contentType != '':
    if contentType == info.gettype():
        contentTypeStatus = 1

# Nacteni html
html = lxml.html.fromstring(request.read())

# Hledani klicoveho slova v tagu meta name="keywords"
result = html.xpath('//meta[@name="keywords"]')
keywords = result[0].get('content').split(',')

if keyword != '':
    for key in keywords:
        if key.strip() == keyword:
            keywordStatus = 1
            break
        #endif
    #endfor
#endif

#xpath Elements
if xpath != '':
    result = html.xpath(xpath)
    xpathElements = len(result)
#endif

# String to find
if stringToFind != '':
    position = html.text_content().find(stringToFind)
    if position >= 0:
        stringToFindStatus = 1
```

```
        #endif
    #endif

    # Predani metriky agentovi
    print 'code=%s' % responseCode
    print 'length=%s' % contentLength
    print 'content-type-status=%s' % contentTypeStatus
    print 'keyword-status=%s' % keywordStatus
    print 'xpath-elements=%s' % xpathElements
    print 'string-to-find-status=%s' % stringToFindStatus

#enddef

# spusteni
getMetric()
```

Výpis 2: Skript http-content.py v jazyce Python

Na začátku jsou definovány proměnné metrik a jsou nastaveny na počáteční hodnoty. Poté dojde k přečtené vstupních parametrů. Tyto parametry jsou označeny vždy dvěma alfanumerickými znaky následovanými znakem rovno „=“ a hodnotou. Toto rozdělené je důležité pro správné přiřazení hodnot metrikám. Ze vstupních parametrů jsou odstraněny počáteční tři znaky a přiřazeny k odpovídajícím proměnným.

Otevře se HTTP spojení na zadanou url adresu pomocí knihovny `urllib2` a její metody `urlopen()`. Výsledek spojení je uložen do proměnné `request`, se kterou se poté dále pracuje.

Nejdříve je nutné získat návratový kód spojené. Toto je provedeno pomocí metody `getcode()`, která vrací HTTP kód odpovědi. Z hlavičky odpovědi je možné také získat informace o velikosti obsahu pomocí metody `info()` a poté `get("Content-Length")`. Také je zde získána informace o typu obsahu. Toto se provede pomocí metody `gettype()`. Provede se porovnání, jestli zadaný typ odpovídá typu z odpovědi. Jestliže ano, poznačí se informace o shodě do proměnné `contentTypeStatus`.

Následuje načtení samotného obsahu což může být HTML kod, vlastní data obrázku a další. Obsah je parsován pomocí knihovny `lxml.html` a její metody `fromstring()` kde je nahrán obsah celé odpovědi. Hledání klíčových slov probíhá tak, že se vyhledá příslušný HTML element, v tomto případě `<meta name="keywords"/>` a jeho atribut `content` se rozdělí pomocí metody `split(',')` do seznamu, kde jsou uložena jednotlivá klíčová slova. Tato klíčová slova se pak porovnávají s hledaným klíčovým slovem, a jestli dojde ke shodě, je toto uloženo do proměnné `keywordStatus`.

Hledání elementů pomocí XPath je realizováno podobně jako hledání klíčových slov. Podle zadané cesty XPath se prohledává obsah a a při shodě je element uložen do seznamu. Zjištění velkosti seznamu se provádí pomocí metody `len()`, která vrací celkový počet položek. Toto číslo udává celkový počet nalezených elementů se zadanou XPath.

Při hledání textového řetězce v obsahuje je použita metoda `text_content`, která zajistí, aby se pracovalo pouze s viditelným textem. Všechny HTML tagy a jejich atributy jsou z obsahu odstraněny. Pro samotné prohledávání je zavolána metody `find()`, jenž vrací při úspěchu číselnou pozici začátku hledaného textu, při neúspěchu vrací -1.

Nakonec je provedeno vytištění dvojic `názevMetriky=hodnota` na standardní výstup odkud je potom pluginem metrika načtena do agenta a dále zpracována.

```
<plugin>
    <property name="version" value="1.0"/>
    <property name="PLUGIN_VERSION" value="4.6.6"/>
    <property name="PLUGIN_NAME" value="http-content"/>

    <service name="HTTP Content">
        <config>
            <option name="script" type="hidden"
            description="http script "
            default="pdk/scripts/http-content.py"/>

            <option name="url"
            description="URL address"
            default="http :// "/>

            <option name="contentType"
            description="Content type"
            optional="true"
            default="text/html"/>

            <option name="keyword"
            description="Keyword to search in META"
            optional="true"
            default=""/>

            <option name="xpath"
            description="Xpath to search"
            optional="true"
            default=""/>

            <option name="findString"
            description="Find visible text string in HTML"
            optional="true"
            default=""/>
        </config>

        < filter  name="template"
        value="exec: file=%script%,args=ur=%url% ct=%contentType% kw=%keyword% xp=%
              xpath% sf=%findString%"/>

        <metric name="Availability"
        template="${template}:Availability "
         indicator="true"/>

        <metric name="Response code"
        category="UTILIZATION"
         indicator="true"
        template="${template}:code"/>

        <metric name="Length"
```

```
            category="THROUGHPUT"
            units="B"
            indicator="true"
            template="${template}:length"/>

            <metric name="Content type"
            category="THROUGHPUT"
            indicator="true"
            template="${template}:content-type-status"/>

            <metric name="Keyword status"
            category="THROUGHPUT"
            indicator="true"
            template="${template}:keyword-status"/>

            <metric name="Number of elements by Xpath"
            category="THROUGHPUT"
            indicator="true"
            template="${template}:xpath-elements"/>

            <metric name="String found"
            category="THROUGHPUT"
            indicator="true"
            template="${template}:string-to-find-status"/>
        </service>
    </plugin>
```

Výpis 3: XML soubor pluginu s deskriptorem

Samotný XML deskriptor je rozdělen na dvě části. V první části je uvedena konfigurace pluginu, která se bude zobrazovat v uživatelském rozhraní Hyperic. Do této konfigurace je zahrnuta i cesta k samotnému skriptu, který sbírá metriku. V tomto případě je cesta nastavena na `pdk/scripts/http-content.py` a má příznak skrytá, takže není přímo v uživatelském rozhraní vidět a ani jí nejde přenastavit.

URL adresa je jediný povinný údaj a musí být zadána, jinak nebude sběr metriky fungovat. Tvar adresy s protokolem ve formátu: `http://doména/jménoStránky.html`.

Je možné zadat Content-Type, který se má objevit v odpovědi. Tato možnost je volitelná díky atributu `optional="true"` a výchozí předvyplněná hodnota ve formuláři je nastavena na `text/html`, což je běžná hodnota pro webovou stránku.

Možnost klíčového slova znamená, že zadané slovo se bude hledat v HTML elementu `<meta name="keywords"/>`. Toto je také volitelná položka a nemá žádnou přednastavenou hodnotu.

Stejné je to u položky `xpath` a `findString`. První slouží k vyhledání a návratu počtu elementů podle zadaného XPath. Druhou metodu je možné použít, když na stránce je text, který je potřeba sledovat, jestli se pokaždé zobrazí.

Zajímavé je použití filtru, jehož konstrukce je trochu složitější. Je rozdělena na několik částí. Jméno filtru je `template`, název slouží pro další použití v rámci samotných metrik. Atribut `value` obsahuje definici spouštění skriptu. První položka `exec:` nám říká, že se jedná o skript, který se budou spouštět. atribut `file` slouží pro načtení skriptu, udává

Obrázek 6: Ukázka grafů pro plugin http-content

se zde cesta ke spustitelnému souboru. V tomto případě je cesta zadána jako proměnná, která dokazuje na konfigurační element `script`. Nejdůležitější v tomto případě jsou vstupní argumenty pro skript, zde zadané v atributu `args`. Jedná se o argumenty, které jsou předány z uživatelského rozhraní skriptu i s jejich tříznakovým prefixem, který slouží pro identifikaci.

Samotné metriky jsou uvedeny v elementech `metric`. Mají hodně společných rysů, ale některý se liší. Všechny používají stejná `template`, který je složen z části z filtru a je k němu přidán název metriky, kterou vrací skript. U metriky se jménem `Length` je uveden atribut `units` s jednotkou `B`, což znamená, že zadané metriky se budou v uživatelském rozhraní zobrazovat v `Byte`.

4.2.2 Otestování pluginu

Plugin byl testován s HTML stránkou na které byly všechny hledané parametry dostupné. Parametry nastavené v uživatelském rozhraní Hyperic pro spuštění testování jsou

následující:

- **URL adresa**: `http://localhost/test_page.html`
- **Content-Type**: text/html
- **Keyword**: test
- **XPath**: //meta
- **Find String**: softwaru

Testovací stránka `test_page.html` má následující podobu:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/
    DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
<head>
< title >test page</title>
<meta http-equiv="content-type" content="application/xhtml+xml; charset=UTF-8" />
<meta name="author" content="Martin Pape" />
<meta name="description" content="Testovac strnky " />
<meta name="keywords" content="test, zkouska, testing," />
<meta name="robots" content="index, follow, noarchive" />
<meta name="googlebot" content="noarchive" />
<link rel="stylesheet" type="text/css" media="screen" href="style.css" />
</head>
  <body>
    <div id="main" class="maindiv">

      <h2>test page</h2>
      <p>
      <strong>Testovani softwaru</strong>.
      </p>
    </div>
  </body>
</html>
```

Výpis 4: Obsah testovací HTML stránky

Na obrázku č. 6 jsou zobrazeny grafy pro jednotlivé metriky pluginu `http-content`. Z grafů je patrné, že v počátečním stavu jsou nalezeny všechny požadované prvky na stránce i v HTTP hlavičce. Návratový kod odpovědi je 200. Typ obsahu (`Contetn-Type`) odpovídá hledanému výrazu `text/html` což je naznačeno hodnotou 1 v grafu. Velikost obsahuje je 2,3kB. Klíčové slovo `test` je nalezeno v:

```
<meta name="keywords" content="test, zkouska, testing," />
```

A proto se v grafu zobrazuje 1. Počet nalezených elementů je celkem 6, jak je patrné z grafu pro XPath. Hledaní textu `softwaru` ve viditelní části stránky je zobrazeno na grafu String found, který vykazuje hodnotu 1.

Po tomto počátečním testu byla provedena editace na souboru `test_page.html`, a odstraněny dva poslední META tagy, změněno klíčové slovo z `test` na `testovaci`

a byla provedena změna vyhledávaného slova `softwaru` na `softwwwaru`. Tímto bylo zapříčiněno, že hledaný viditelný text nebyl nalezen, což je naznačeno na grafu hodnotou 0, prohledávání elementů pomocí XPath objevilo jen 4 elementy a klíčové slovo `test` nebylo nalezeno, protože je v grafu zobrazena 0.

Ještě bylo provedeno několik samostatných testů, aby se prokázalo že funguje správně vyhledání typu obsahu. Testy probíhali na různých typech obrázků, textů a audiovizuálním obsahu.

5 Závěr

Cílem této práce bylo navrhnout a implementovat rozšiřující pluginy pro monitorovací systém Hyperic HQ. Jeden plugin se měl zabývat sběrem dat z databázové tabulky a kontrolou jestli data přibývají. Druhý plugin měl za úkol shromaždovat informace o HTTP připojení na zadanou URL adresu.

Plugin pro sběr dat z databáze byl implementován pomocí podpůrných tříd Hyperic, což se ukázalo jako velice přínosné pro vytvoření celého pluginu sestávajícího pouze z XML souboru za velice krátkou dobu. Omezením však je nízká možnost rozšíření funkcionality a je potřeba dodržovat požadavky podpůrné třídy. Při tvorbě tohoto pluginu bylo potřeba provést implementaci pro každý typ podporované databáze samostatně. Také bylo potřeba zajistit korektní zobrazení databázového typu `datetime` v uživatelském rozhraní Hyperic. Bylo proto potřeba převést tento časový údaj na typ `timestam`, který je poté zobrazován bez problémů. Tento plugin by se v budoucnu mohl dát ještě rozšířit, ale považuji jeho funkcionaluti pro dostačující pro zadaný úkol.

Plugin pro sledování stavu URL dle vráceného obsahu byl implementován pomocí skriptovacího jazyka Python, který dovoluje provádět složité akce ve srozumitelné formě zápisu. Má velké množství rozšiřujících knihoven a je dostupný pro všechny platformy. Hyperic HQ dovoluje při implementaci pluginů využít vlastní skripty, ale je potřeba striktně dodržet formát předávání metrik. Tento formát je vytištěn na standardní výstup, odkud je předán HQ agentovi. V zadání pro tento plugin bylo uvedeno, že má velikost obsahu, mime-type a obsažená klíčová slova. K těmto funkcím byly přidány ještě další pro monitorování návratového kodu odpovědi, zjištění počtu elementů pomocí zadané XPath a vyhledání viditelného textu na stránce. Pro správnou funkci skriptu je potřeba mít nainstalovanou knihovnu `lxml` jejíž instalace se provede snadno pomocí příkazu: `pip install lxml`. Tato knihovna je použita pro parsování HTML obsahu a hledání pomocí XPath. Jako největší problém při implementaci se ukázalo správně napojení XML deskriptoru na vlastní skript, které je realizované pomocí předávání několika vstupních parametrů a musí být uvedeno ve správné formě v atributu `template` v XML deskriptoru. Tento plugin je ještě možné určitě dále rozšiřovat přidáním požadovaných funkcí. Záleží už potom na konkrétních požadavcích.

6 Reference

[1] Erich Gamma Richard Helm, Ralph Johnson, John Vlissides (Gang of Four) *Návrh programů pomocí vzorců*. Grada. Praha 2003. ISBN 8024703025

[2] *Hyperic Application System Monitoring* [online]. [cit. 2012-10-05]. Dostupné z: <http://sourceforge.net/projects/hyperic-hq>

[3] *Hyperic HQ Tour* [online]. 25.2.2009 [cit. 2013-01-05]. Dostupné z: <http://support.hyperic.com/download/attachments/59375779/HypericHQTour.pdf>

[4] *Hyperic HQ 3.0 Open Source Test Ride Guide* [online]. Březen 2007 [cit. 2013-03-09]. Dostupné z: <http://docs.huihoo.com/hyperic-hq/hyperic-hq-3.0-test-ride-guide.pdf>

[5] *Dva nejlepší bezplatné nástroje pro správu IT* [online]. Duben 2013 [cit. 2013-04-15]. Dostupné z: <http://www.businessit.cz/cz/dva-nejlepsi-bezplatne-nastroje-pro-spravu-it.php>

[6] *vFabric Hyperic Overview* [online]. 2012 [cit. 2013-04-18]. Dostupné z: <http://pubs.vmware.com/vfabric52/topic/com.vmware.ICbase/PDF/vfabric-hyperic-overview-4.6.6.pdf>

[7] *vFabric Hyperic Product Plug-in Development* [online]. 2012 [cit. 2013-04-18]. Dostupné z: <http://pubs.vmware.com/vfabric52/topic/com.vmware.ICbase/PDF/vfabric-hyperic-plugindev-4.6.6.pdf>

Seznam tabulek

Seznam obrázků